ISC High Performance
The HPC Event.

ISC15 Workshop for Intel® Xeon Phi™ Processors

Language Impact on Vectorization: Vector Programming in Fortran

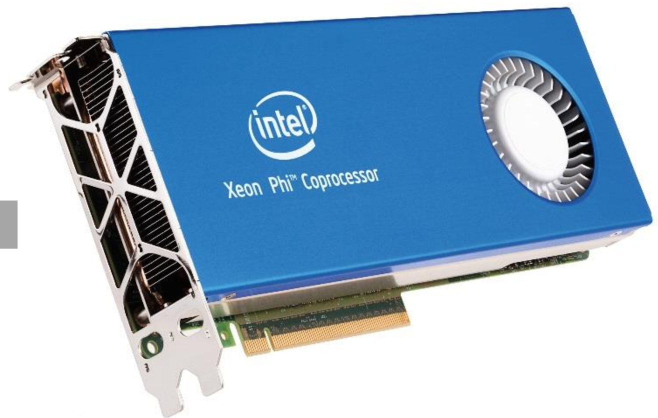
Florian Wende

Zuse Institute Berlin

Synopsis

SIMD parallelism is one key mechanism to be addressed when writing code for modern multi- and manycore processors

AVX512



How SIMD-friendly is my code?
How to adapt and modernize?
How to avoid compiler specific modifications?

Performance?

autovectorizer
vs.
explicit vector programming (compiler directives)
vs.
manual vector programming (assembly, intrinsics)

Vector Programming in Fortran

Language constructs already enabling SIMD vectorization

- Array notation: `a(:) = 2.0 * b(:);   call f(s,a(:),b(:))`
- `where` construct for conditional array operations

```
where (a > 0.0)
   a = log(a)
elsewhere
   a = 0.0
end where
```

> 👍 Clear code: compiler can do SIMD
> 👎 You do not get the mask!
> 👎 Nested `where` seems not so efficient!

- `forall` construct to define per-element sequence of operations

```
forall (i=1:16)
   a(i) = 2.0 * b(i)
   a(i) = exp(a(i))
end forall
```

Vector Programming in Fortran – Explicit++

More control with vector data types: define them yourself

- Define SIMD width: `#define SIMDWIDTH 8` (e.g. for IMCI, 64-bit words)
- Vectors containing 64-bit floating point elements + vector mask

```
type::simd_real8
real*8::x(0:SIMDWIDTH-1)
end type simd_real8

type::simd_mask8
logical::x(0:SIMDWIDTH-1)
end type simd_mask8
```

- Compile code with `-align`*`n`*`byte` (Intel specific) or use compiler directive `!dir$ attributes align` (Intel specific) in code

Vector Programming in Fortran – Explicit++

How to adapt the code?

```
...
do i=1,n
  call foo(x1(i),x2(i),y(i))
enddo
...
```

foo refers to the loop body or a SIMD function

```
type(simd_real8)::buffer_x1,buffer_x2,buffer_y
type(simd_mask8)::m
...
do i=1,n,SIMDWIDTH
!$omp simd
  do ii=0,SIMDWIDTH-1
    if ((i+ii).le.n)
      m%x(ii)=.true.
      buffer_x1%x(ii)=x1(i+ii)
      buffer_x2%x(ii)=x2(i+ii)
    else
      m%x(ii)=.false.
    endif
  enddo
  call foo_simd(buffer_x1,buffer_x2,buffer_y,m)
!$omp simd
  do ii=0,SIMDWIDTH-1
    if (m%x(ii)) y(i+ii)=buffer_y%x(ii)
  enddo
enddo
...
```

Load data into vector buffers + create mask

SIMD function call

Store content of vector buffers to output array

Vector Programming in Fortran – Explicit++

```
subroutine foo(x1,x2,y)
real*8::x,y,a
a=x1*x2
if (a<=0.0)
  y=0.0
  return
endif
a=sqrt(a)
y=log(a)
end subroutine foo
```

Early return

Similar for

- conditional code blocks
- `while` loops on SIMD lanes
- nested SIMD-enabled functions

```
subroutine foo_simd(x1,x2,y,m0)
type(simd_real8)::x1,x2,y,a
type(simd_mask8)::m0,m1
integer::i
logical::true_for_any=.false.
!$omp simd reduction(.or.:true_for_any)
do i=0,SIMDWIDTH-1
  a%x(i)=x1%x(i)*x2%x(i)
  if (m0%x(i).and.a%x(i)>0.0) then
    m1%x(i)=.true.
    true_for_any=.true.
  else
    m1%x(i)=.false.
    y%x(i)=0.0
  endif
enddo
if (.not.true_for_any) return
!$omp simd
do i=0,SIMDWIDTH-1
  if (m1%x(i)) then
    a%x(i)=sqrt(a%x(i))
    y%x(i)=log(a%x(i))
  endif
enddo
end subroutine foo_simd
```

Mask `m0` from caller

Return if all SIMD lanes are inactive

Vector Programming in Fortran – Explicit++

Performance of early return on a Haswell System and Xeon Phi:

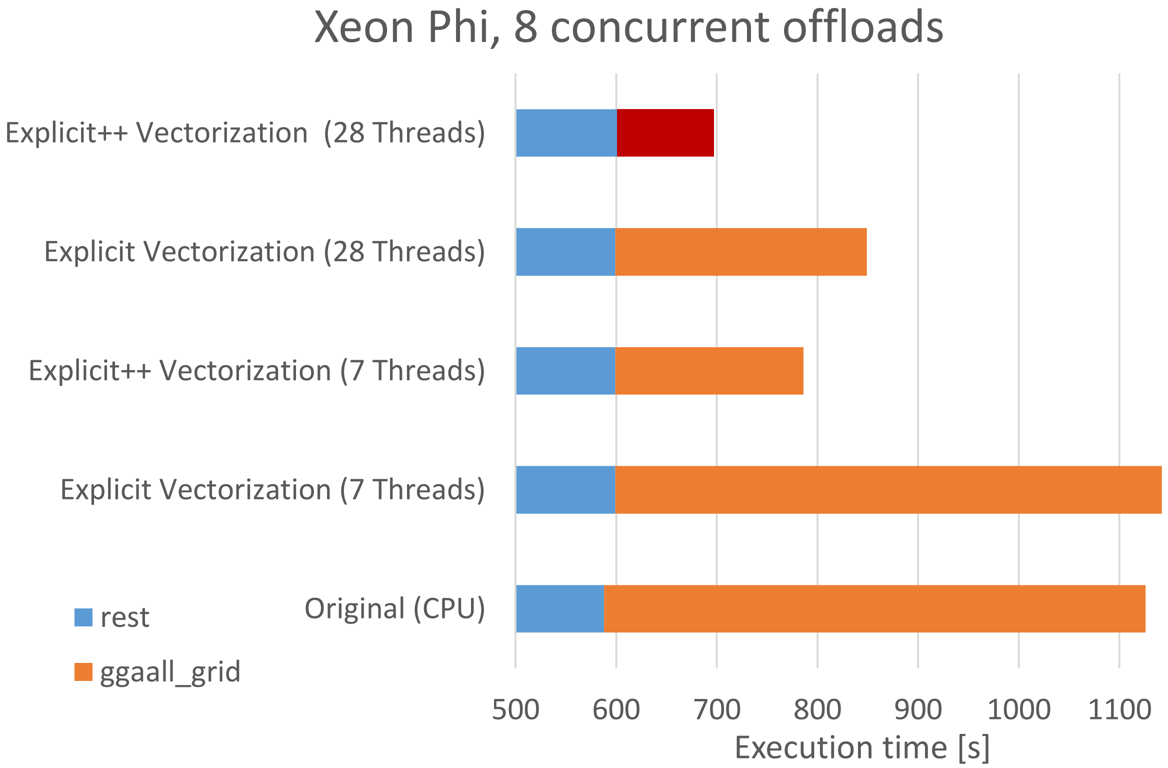
- random input: x1() and x2() from [-100.0,+100.0]
- only half the number of SIMD lanes active on average

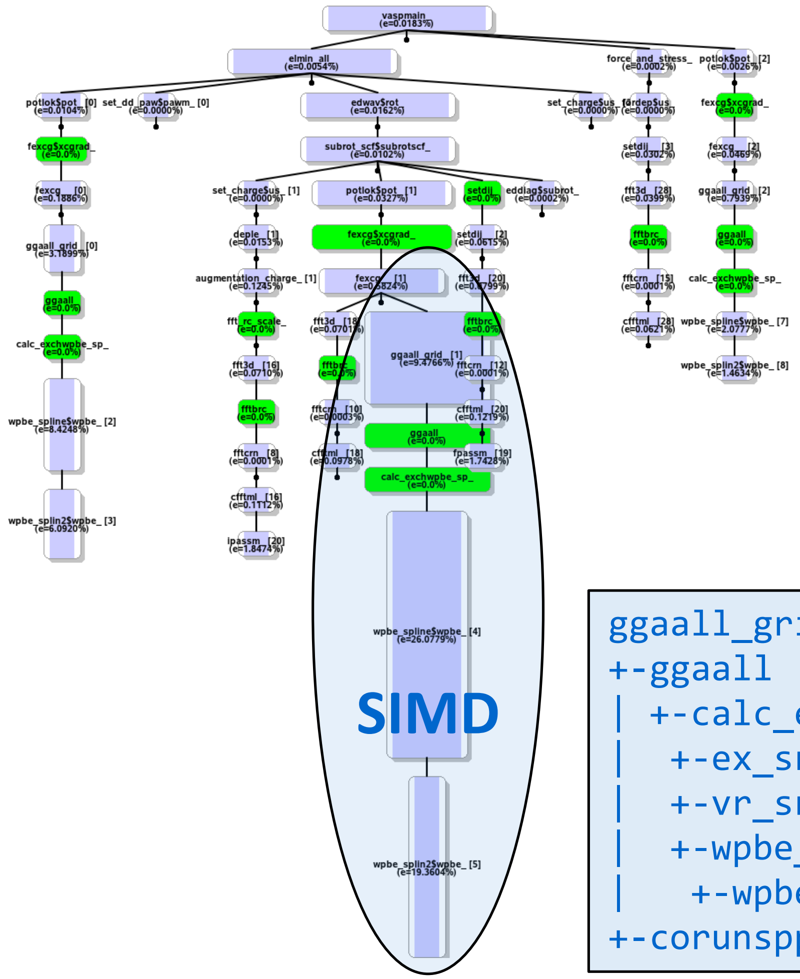
	Haswell, AVX2*	**Xeon Phi, IMCI***
Scalar	150ms	1310ms
Explicit vectorization	136ms	985ms
Explicit++ vectorization	81ms	575ms
SIMD intrinsics (C)	67ms	395ms

** Just one thread used for execution on both CPU and Xeon Phi*

Vector Programming in Fortran – VASP

We use explicit(++) vectorization in Fortran to vectorize along the calling tree






```
ggaall_grid
+-ggaall
| +-calc_exchwpbe_sp
|   +-ex_sr
|   +-vr_sr
|   +-wpbe_spline
|    +-wpbe_splin2
+-corunsppbe
```

Thank You

Vector Programming in Fortran – Explicit++

Performance of early return on a Haswell System and Xeon Phi:

- random input: x1() and x2() from [-100.0,+100.0]
- only half the number of SIMD lanes active on average

	Haswell, AVX2*	**Xeon Phi, IMCI***
Scalar	150ms	1310ms
Explicit vectorization	136ms	985ms
Explicit++ vectorization	81ms	575ms
SIMD intrinsics (C)	67ms	395ms
SIMD intrinsics (C call from Fortran, entire function)	73ms	-
SIMD intrinsics (C call from Fortran, instruction level)	90ms	-

** Just one thread used for execution on both CPU and Xeon Phi*